Universidade federal São João Del Rei
Campus: Tancredo de Almeida Neves
Departamento de Ciência da Computação
Algoritmo e Estrutura de Dados III

Trabalho Prático 1: HIPERCAMPO

Alunos:

Luiz Felipe, 172050101
Sidney Júnior, 172050095

Professor: LEONARDO ROCHA

Fevereiro
2021

Conteúdo

1 Introdução **1**

2 Especificações **1**

3 Idealização **2**

4 O Algoritmo **3**
4.1 Análise 3
4.1.1 As validações 4

5 Resultados **4**
5.0.1 Máquina 5
5.1 Os arquivos de teste 5
5.1.1 Gráfico 5

6 Complexidade **7**

7 Conclusão **8**

1 Introdução

Este trabalho prático proposto compreende o problema de Hipercampos. E uma das possíveis formas para a obtenção da resolução final do mesmo.

Primeiramente, o desafio desse trabalho não é apenas um tipico problema visto na computação, no qual se resolveria com um algoritmo simples. O desafio proposto pode ser resolvido com ajuda de diversos algoritmos. No entanto, dentre esses algoritmos o problema em questão é encontrar uma de melhor resolução no qual a sua complexidade seja boa, independe dos números de entradas.

O hipercampos parte da utilização de um arquivo de entrada, em busca de encontrar as possíveis combinações de pontos que não se colidem.

Adquirindo as seguintes informações, usando o sistema cartesiano de planos X e Y, assumimos duas âncoras $A = (X_A, 0)$ e $B = (X_B, 0)$ na qual, possuímos um conjunto de Ponto com N pontos ordenados por (X, Y), sendo que $X > 0$ e $Y > 0$. Os pontos são conectados às âncoras por segmentos de reta, pertencentes a Ponto, que partem de cada ponto (v, A) e (v, B).

Com a utilização de um algoritmo de otimização, tentamos obter o máximo de combinações possíveis na qual as retas traçadas entres os possíveis pontos, nunca se colidem entre si.

2 Especificações

Dada a documentação do trabalho pode-se obter as seguintes especificações para a mostra desse problema:

- Pode-se afirmar que as duas âncoras citadas estão afixadas ao eixo X, e consequentemente ao eixo Y, na qual o valor máximo para a última âncora tem o limite de até 10^4 unidades em relação ao ponto 0, logo concluindo que $(0 < X_A < X_B \leq 10^4)$.
- Temos o conjunto N de pontos coordenados entre (X,Y), nele sabemos que $N > 0$ e $N < 100$ logo temos que nosso $(0 < N \leq 100)$.
- Não existe nenhum ponto que se colidem ou ponto coincidentes.
- Dado U e V, levando em consideração o tópico acima, temos que u e v são distintos , tais como A,u,v ou B,u,v sejam colineares.

3 Idealização

Entre as varias possibilidades de resolver o problema do hipercampo, uma delas inicialmente seria com a utilização de um Algorítimo de Força Bruta*(busca por exaustão)*, pois nele tentamos todas as entradas do arquivo, sendo possível verificar todas as combinações dadas. Porem como visto dentro de sala de aula, um algoritmo de força bruta nem sempre é a melhor escolha, como por exemplo neste problema, o algoritmo tomaria uma forma de ordem de complexidade fatorial sendo esta $O(n!)$, pensando num arquivo com entradas de $N = 100$, o mesmo se torna inviável pois para a execução demandaria muito tempo.

Levando em consideração a desvantagem do algoritmo de força bruta, uma outra solução seria usar as Técnicas de Programação Dinâmica, que consiste *"assim como o método de divisão e conquista, resolve problemas combinando as soluções para subproblemas. Um algoritmo de programação dinâmica resolve cada sub subproblema só uma vez e depois grava sua resposta em uma tabela, evitando assim, o trabalho de recalcular a resposta toda vez que resolver cada sub subproblema."* [1]

Sabendo que o problema proposto tem varias soluções possíveis, para a resolução usando Programação Dinâmica*(***problemas de otimização***)*, precisamos seguir algumas regras definidas, estas regras são em quatro etapas sendo elas:

1. Caracterizar a estrutura de uma solução ótima.
2. Definir recursivamente o valor de uma solução ótima.
3. Calcular o valor de uma solução ótima, normalmente de baixo para cima.
4. Construir uma solução ótima com as informações calculadas.

Dada toda complexidade do problema recorremos ao uso de **geometria analítica**, para nos ajudar na compressão do mesmo, Tendo em mente o uso da **Equação Fundamental da Reta** na qual, uma reta *" é obtida por um ponto pertencente a essa reta mais o seu coeficiente angular, ficando sempre em função de outro ponto."* [2]

Com todo os pontos levantado, partimos para formulação dos pontos no GEOGEBRA, o mesmo sendo usado para gerar alguns pontos validos e consequentemente gerando alguns arquivos de testes com os resultados já esperado, com todo os matérias e as pesquisas levantadas podemos parti para a escrita do código, testagem, correção de erros e melhoria, assim podendo chegar ao algoritmo utilizado neste trabalho.

4 O Algoritmo

Primeiramente, para elaboração do programa foi necessário receber dois parâmetros, com a ajuda da primitiva **getopt**, que respectivamente, recebe o nome nomes dos arquivos de entrada e saída inserida pelo usuário.

Em seguida, usa-se uma TAD *(Tipo Abstrato de Dados) sendo eles Arquivos.h e Arquivos.c*, para realizar a verificação e leitura dos dados do arquivo de entrada.

Levando em consideração o eixo X, apos a abertura e leitura do arquivo em seguida armazenados, os pontos são organizados por meio do algoritmo de ordenação Quicksort, pois o mesmo é um algoritmo recursivo que utiliza a estratégia da **divisão e conquista**. A escolha do Quicksort se deu pois, seu pior caso tem um tempo de execução em $\ominus(n^2)$ e se bem implementado o custo é de $\ominus$(n log n), tendo a desvantagem de não ser um algoritmo estável.

Logo apos a esse processo, o algoritmo faz as verificações para que possamos obter a quantidade máxima de pontos, com ajuda do algoritmo de programação dinâmica em relação ao arquivo de entrada utilizado, ou seja, a implementação dos resultados obtidos foi feita de forma dinâmica.

4.1 Análise

Para que o hipercampo seja valido, são feitas algumas validações ate que se chegue na resposta final satisfatória, seguindo uma logica, tendo a seguinte ideia:

Depois de toda as validação dos arquivos passados pelo usuário, como também, seus pontos terem sidos validados, seguindo as especificações (2 Item 1), o mesmo são armazenado para que sejam ordenados pelo *Quicksort*. Levando em consideração a altura, parti-se do menor ponto até chegar ao valor desejado. A cada verificação dos pontos, descobrimos quantos pontos pode pertencer aquele seguimento Z = (X,a) - (Y,a), sendo assim armazenado num vetor.

Uma grande parte do trabalho se inicia a partir desse vetor ordenado pelo Quicksort, logo em seguida de forma dinâmica, validamos o maior ponto, fazendo uma checagem com os pontos menores que ele, para assim saber se os pontos em questão pertence aqueles seguimento testado. Neste contexto assim que encontrado o ponto que comporta o maior numero de pontos dentro dele, ele e adicionado ao vetor[i] e logo em sequencia soma mais na posição i, que passa a ser seu valor atual, assim quando terminado esse processo com todos os pontos possíveis, faz-se uma comparação com uma variável na qual armazena o maior valor encontrado, todos os valores são armazenado nas suas posição, ate que todos os pontos sejam conferidos. Só então que o

resultado e gravado no arquivo de saída.

4.1.1 As validações

A dificuldade de se pensar esse problema se dá a partir do segundo ponto, pois quando temos mais de um ponto, temos que checar se aquele reta que queremos inserir não colide com alguma outra que já esta dentro das retas válido, considerando que o nosso programa assume na primeira vez de interação na qual o maior ponto é o 1, pois não deve haver nenhum ponto maior que ele naquela posição.

A partir do segundo ponto, todos os valores anteriores ao mesmo deve ser checado, para saber se não existe colisão entre as retas já válidas e o ancora em questão,e assim podendo assumir que aquele reta pode ser inserido naquela altura sem colisão com as demais, logo garantindo que qualquer ponto dentro da reta em questão esta dentro e que não se colidem.

Toda a validação e possível com a ajuda da *equação fundamental da reta* [2], visto em geometria analítica, seguindo a ideia a seguir:

- Na reta da esquerda, sua formação é dada pelo ponto atual até a sua âncora da esquerda ($\hat{a}ncora_A$), e a âncora da direita ($\hat{a}ncora_B$) é dada pelo ponto anterior ao que está sendo verificado. Se essas retas não colidirem, que dizer que está correto.

- Na reta da direita, sua formação é feita pelo ponto atual até a âncora da direita ($\hat{a}ncora_B$) e a reta da esquerda tem sua formação dado pelo ponto anterior analisado e a ancora da esquerda($\hat{a}ncora_A$). Se essas retas não colidirem, que dizer que está correto.

Levando em consideração os item acima, se ambos forem corretos, quer dizer, não se colidem, podemos afirmar que aquele ponto em questão está dentro das retas testadas anteriormente. Consequentemente, assim que um valor maior é encontrado, o valor do ponto antigo entrega o lugar ao valor em questão.

Depois de todas as análises citadas acima, fazemos uma última comparação, que é com o valor máximo atual, caso o valor em questão seja menor que ele esse valor e descartado, se for maior substituímos o valor anterior para o valor atual.

5 Resultados

Os teste neste período emergencial dependia das maquinas dos membros do trabalho, sendo testados com a ajuda do WSL no Windows, na emula

o sistema Ubuntu 20.04 e na mesma maquina que tinha como o sistema principal como Ubuntu 20.04.

Os resultados obtido pelo membros, com os aquivos de teste criados no GEOGEBRA e os arquivos criados com um gerador de pontos aleatórios, além de bem sucedidos,tanto usando o WSL do Windows ou com o próprio sistema Ubuntu 20.04 usado como sistema principal, fora satisfatório, independentemente se era um arquivo com uma entrada pequena ou ate mesmo com arquivos com a quantidade grande de entrada.

5.0.1 Máquina

Para os testes de tempo, foi utilizado a função *getrusage e gettimeofday*, e os testes foram feitos em um dos computadores pessoais dos membros do trabalho. C as seguintes configurações:

Configurações

- **Sistema operacional:** Windows 10 - usando WSL(Bash via terminal com Ubuntu 20.04) - x64 e Ubuntu 20.04 - 64bits
- **Processador:** Intel(R) Core(TM) I7-9750H CPU 2.60GHz
- **Memoria RAM:** 8 GB

5.1 Os arquivos de teste

Para verificar a eficacia do projeto foram testados 20 arquivos, em que, cada um desses arquivos continham uma quantidade de pontos distintos, gerados aleatoriamente, sendo salvos nas suas respectivas saídas após a execução do projeto. Recolhemos os seguintes dados dos arquivos e em seguida montamos uma tabela, com os resultados usando gettimeofday e getrusage e logo em seguida um gráfico que contem *numero_de_pontos X Tempo*.

5.1.1 Gráfico

Os dados as tabelas citada no (6 tópico 6.2 e 6.3), conseguimos gerar os seguintes gráficos:

Entradas	Tempo Gettimeofday (µs)	Tempo Getruser (1.e-6 s)
10	62578 µs	0.015625 s
20	72962 µs	0.015625 s
30	143176 µs	0.015625 s
40	128040 µs	0.015625 s
50	65313 µs	0.015625 s
100	127527 µs	0.015625 s
200	128586 µs	0.015625 s
300	129875 µs	0.015625 s
400	131516 µs	0.015625 s
500	71319 µs	0.015625 s
1000	153583 µs	0.031250s
2000	127051 µs	0.046875s
3000	193991 µs	0.125000s
4000	279562 µs	0.218750s
5000	401934 µs	0.343750s
10000	1389205 µs	1.328125s
50000	33663878 µs	33.593750s
100000	133543030 µs	133.406250s
200000	3520882063 µs	3518.859375s
100000	13699204739 µs	13693.296875s

Tabela 1: Medida de tempo usando o WSl do Windows para executar o Hipercampo usando o Ubuntu 20.04.

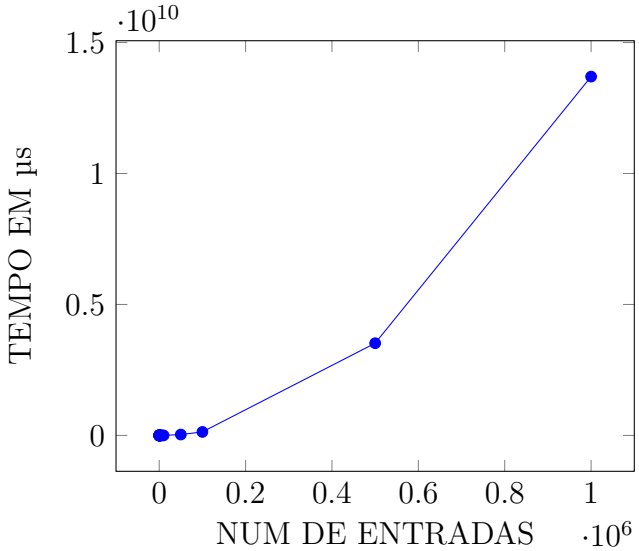

Gráfico 1: gerado com os dados da função Gettimeofday mostrada acima.

Os valores testado vai de uma pequena quantidade de numero do aquivo de entrada ate mesmo um arquivo com entradas muito grandes.

Com isso é possíveis vemos que o programa tem um custo dependente da entrada inserida pelo usuário, e essa mudança ocorre de uma forma quase constante ao adicionar ou remover entradas

6 Complexidade

Para que possamos fazer uma analise de complexidade de forma mais tranquila, temos que desconsiderar o custo das leituras de arquivo, a alocação de memoria e também a escrita no arquivo, dados esses ponto a analise de complexidade é feita da seguinte forma:

Dado um N, sendo ele o número de pontos possíveis no hipercampo, em que, levando em consideração as trocas feitas por ele para que possamos analisar.

Com a ajuda do Quicksort, sabemos que no pior caso ele tem sua complexidade em $\ominus(n^2)$ como visto na **secção 4**. Em sequencia, temos a função **Colisao()**, na qual tem a instrução de comparar dois seguimentos de reta, com isso, tendo um custo de $\ominus(1)$.

Partimos agora para a função de **validacao()**, que tem como instrução chamar a função de colisão citada acima, com isso, tornando a complexidade desta função tendo um $MAXIMO(\ominus(1), \ominus(1))$ sendo assim $Maximo = \ominus(1)$.

Seguindo para a função **soluciona()**, passamos um vetor ordenado, a quantidade de pontos do arquivo, e mais dois parâmetros, que contem a logica de funcionamento do programa. Nela percorremos o vetor com N entradas, e logo em seguida chamamos nossa **validacao()** quem contém uma quantidade de i , na qual, para todo os valores de i, $(0 < i < n)$, tendo sua complexidade sendo:

$$\boxed{\sum_{i=2}^{\mathrm{N}} i}$$

e sua ordem de complexidade é $\ominus(n^2)$. o algoritmo desenvolvido no final tem sua ordem de complexidade $MAXIMO(\ominus(n^2), \ominus(n^2), \ominus(1)$, tendo assim uma complexidade final de $\ominus(n^2)$

7 Conclusão

Analisando a complexidade inicial do problema proposto para este trabalho de hipercampos, os testes utilizando busca por forca bruta se mostrariam muito ineficientes,pois essa técnica é simples de ser implementada porem é ineficiente em quesito de custo computacional, inviabilizando seu uso em problemas desse tipo.

O uso de programação dinâmica trás vantagens, uma delas é a possibilidade de se usar algoritmos de programação em que armazenamos os resultados dos testes anteriores, facilitando a próxima checagem dos dados. Tornando assim essa técnica mais eficaz para o problema proposto, como observado na tabela 1 do item 6.2.1 a relação entre o tamanho do arquivo de entrada e o tempo de espera são proporcionais.

Visto a complexidade do problema, o uso da equação fundamental da reta estudada em geometria analítica torna-se um facilitador para se idealizar um modelo de partida para a criação do algotimo que o representa-se. Logo foi de extrema importância para facilitar a visualização do problema que de inicio se mostrava muito complexo.

Referências

[1] T.H. Cormen, C.E. Leiserson, and R.L.R.E.C. Stein. *Algoritmos: teoria e prática*. ELSEVIER EDITORA, 2012.

[2] Danielle de Miranda. Equação fundamental da reta - mundo educação.